Machine Learning

Homework 6

Due: November 14, 2008 (midnight)

No late homeworks will be accepted.

Most approaches to machine learning assume the knowledge to be learned can be expressed as a set of attribute-value pairs, i.e., an entity and its properties. In fact, Weka [8] assumes input data to be of this form. However, much richer knowledge can be found in the relationships between multiple entities. For example, computational chemistry applications look for patterns in the relationships between atoms and molecules, not just patterns in the properties of a set of atoms. Such analysis can discover chemical structures (e.g., a benzene ring); whereas, attribute-value-based learning methods cannot express such knowledge. Relational learning is an area within machine learning that specifically addresses the challenge of learning relational knowledge [4].

There are basically two approaches to learning relational knowledge, and they center around the two main representations for relational knowledge: first-order logic and graphs. First-order logic is vastly more expressive than propositional logic, which is the representation of attribute-value-based learning methods. Several methods exist for learning relational knowledge in the form of first-order logic, and these methods fall under the area of Inductive Logic Programming (ILP) [6]. Graphs (i.e., a collection of nodes and links between nodes) are not quite as expressive as first-order logic, but they are an efficient and intuitive representation for relational knowledge. Several methods exist for learning relational knowledge in the form of graphs, and these methods fall under the area of Graph-based Data Mining or Graph-based Relational Learning [5].

This assignment will explore the challenge of learning relational knowledge by experimenting with two relational learning methods: the PROGOL logic-based relational learner and the SUBDUE graph-based relational learner. Throughout this assignment, there are items marked as "*Deliverable*" that are the expected outputs of the assignment. Write up the answers to the questions in a document, include it and all the input/output files into a zip file, and email it to me (holder@wsu.edu) by the above deadline.

1 Logic-based Relational Learner: PROGOL

For our first foray into relational learning, we will use the PROGOL logic-based relational learning system [7]. PROGOL is an inductive logic programming (ILP) system that uses inverse entailment, guided by a user-defined mode declaration, to search for a logic theory that entails the correct classification of the training examples. A mode declaration is a constraint which imposes restrictions on the atoms and their arguments appearing in a clause of the learned theory by (1) determining which atoms can occur in the head and the body of the clause, (2) determining which arguments can be input variables, output variables or constants, and (3) determining the number of alternative solutions for instantiating the atom. PROGOL first computes the most specific clause *S* which covers a seed example and then uses A* search, guided by a compression and accuracy measure, to generalize *S* in order to cover more positive

examples. PROGOL can also accept arbitrary Prolog clauses as background knowledge that can be referred to by the learned theory.

1.1 Initial Test with PROGOL

First, we will download, install and test PROGOL. The main download site for PROGOL is given in [1]. After unpacking the distribution, go to the `source` directory and type `make` to compile `progol`. While this version should compile on most UNIX systems using the Gnu `gcc` compiler, you may have to add the `–m32` option to the `CFLAGS` variable in the `source/Makefile`, as the code has problems on 64-bit architectures. You should then read through the tutorial in `man/tutorial4.4.ps`. The tutorial describes the execution of PROGOL on the `examples4.2/aunt.pl` file.

Deliverable 1: Provide the output produced by PROGOL on the `aunt.pl` example and give an English description of the theory learned by PROGOL.

1.2 Teaching PROGOL the Concept of a Benzene Ring

Now that you have some experience using PROGOL, let's construct our own input file to teach PROGOL the concept of a benzene ring (see Figure 1). A benzene ring is a ring of six carbon (C) atoms connected by alternating single and double bonds. We will ignore the hydrogen (H) atoms.

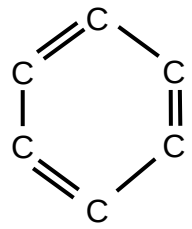

Figure 1: Benzene ring.

We will first attempt to teach PROGOL the concept of benzene by showing it only positive examples of the molecule based on two background predicates: `single_bond(C1,C2)` and `double_bond(C1,C2)`. The specific predicate to learn will be `benzene_ring(C1,C2,C3,C4,C5,C6)`. We can assume the arguments to these predicates are all of type `carbon`.

Deliverable 2: Based on the above scenario, construct a PROGOL input file with ten positive examples of a benzene ring. Following the `aunt.pl` example file, your input file will need to set the `posonly` setting, have mode declarations for the `benzene_ring`, `single_bond`, and `double_bond` predicates, have `carbon` type predicates for each carbon involved in the examples, have background knowledge about the single and double bonds involved in the ten different benzene rings, and then the ten benzene example predicates. Provide your input file and corresponding PROGOL output file. *Hint*: See the following discussion.

Most likely, despite the use of differing representations and number of examples, PROGOL will not produce the expected theory, i.e., a conjunction of three single bond predicates and three

double bond predicates with appropriately common variables. However, the theory that PROGOL does produce is correct, i.e., it correctly describes (or "covers") all the examples. For instance, PROGOL is likely to produce the following theory:

```
benzene_ring(A,B,C,D,E,F) :- single_bond(A,B).
```

The above theory is correct, i.e., every benzene ring contains a single bond between two carbon atoms.

So, how can we get PROGOL to produce the correct theory? We need to give PROGOL some negative examples of a benzene ring, e.g., a benzene ring with one bond missing. Let's give this a try.

Deliverable 3: Starting with the above input file with ten positive examples of a benzene ring, add six negative examples. Each negative example will be a benzene ring, but with one of the bonds missing (a different bond missing for each of the six negative examples). You will also need to set the 'c' parameter to 6 (use '`:- set(c,6)?`'), which tells PROGOL there can be six predicates in the clause. Note: PROGOL's running time is exponential in 'c', so the default is 3. Provide your input file and corresponding PROGOL output file.

If all goes well, PROGOL should learn the following correct theory for the benzene ring.

```
benzene_ring(A,B,C,D,E,F) :-
     single_bond(A,B),
     single_bond(C,D),
     single_bond(E,F),
     double_bond(A,F),
     double_bond(B,C),
     double_bond(D,E).
```

Now that we have some experience learning relational concepts using a logical representation, let's try some similar tasks using a graph representation.

2 Graph-based Relational Learner: SUBDUE

The SUBDUE [3] graph-based relational learning system accepts input as a directed graph with labels on vertices and edges. SUBDUE uses a beam search to identify a subgraph (or substructure) of the input graph that best compresses the input graph. The initial state of the search is the set of substructures consisting of all uniquely labeled vertices. The only operator of the search is the Extend Substructure operator. As its name suggests, it extends a substructure in all possible ways by a single edge. SUBDUE's search is guided by the minimum description length (MDL) principle, which seeks to minimize the description length of the entire data set. The evaluation heuristic based on the MDL principle assumes that the best substructure is the one that minimizes the description length of the input graph when compressed by the substructure. The description length of the substructure *S* given the input graph *G* is calculated as $DL(G,S) = DL(S)+DL(G|S)$, where $DL(S)$ is the description length of the substructure, and DL(G|S) is the description length of the input graph compressed by the substructure. Subdue

seeks a substructure *S* that minimizes *DL(G,S)*. The search terminates upon reaching a user-specified limit on the number of substructures extended, or upon exhaustion of the search space.

2.1 Initial Test with SUBDUE

First, we need to download, install and test SUBDUE. The main download site for SUBDUE is given in [2]. After unpacking the distribution, follow the instructions in the `README` file to install SUBDUE. You should then read through the user's manual in `docs/UsersManual_1.4.pdf`. The manual describes the execution of SUBDUE on the `graphs/sample.g` file.

Deliverable 4: Provide the output produced by SUBDUE on the `sample.g` example and give an English description of the best substructure learned by SUBDUE.

2.2 Teaching SUBDUE the Concept of a Benzene Ring

Now that you have some experience using SUBDUE, let's construct our own input file to teach SUBDUE the concept of a benzene ring (see Figure 1). Recall that a benzene ring is a ring of six carbon (C) atoms connected by alternating single and double bonds. Again, we will ignore the hydrogen (H) atoms.

We will first attempt to teach SUBDUE the concept of benzene by showing it only positive examples of the molecule based on two relations: `single_bond` and `double_bond`.

Deliverable 5: Based on the above scenario, construct a SUBDUE input file with three positive examples of a benzene ring. Following the `sample.g` example file, your input file will need to define each benzene-ring graph by defining six vertices labeled `C` for the six carbon atoms, and then six undirected edges, three labeled `single_bond` and three labeled `double_bond`. Since we need to define three such graphs, we could continue with vertices 7-12 and 13-18, but SUBDUE allows you to provide multiple positive examples, each starting from vertex 1, by including the line "`XP`" just before each example. Provide your input file and the corresponding SUBDUE output file.

If all goes well, SUBDUE should have learned the following correct graph structure for a benzene ring as the best substructure.

```
v 1 C
v 2 C
v 3 C
v 4 C
v 5 C
v 6 C
u 1 2 single_bond
u 3 4 single_bond
u 5 6 single_bond
u 2 3 double_bond
```

```
u 4 5 double_bond
u 6 1 double_bond
```

SUBDUE is able to learn the benzene-ring structure using only positive examples (unlike PROGOL), because SUBDUE uses a different heuristic to guide the learning; namely, compression. SUBDUE searches for the structure that maximally compresses the input graphs. In this case, once SUBDUE has enough positive examples of a benzene ring, then this same structure will afford maximal compression when each example is compressed to a single vertex. PROGOL, on the other hand, is guided by the goal of correctly covering the positive examples, and so chooses the smallest subpart contained in every example as its hypothesis.

As an exercise, let's go ahead and give SUBDUE some negative examples of the benzene ring and see what effect they have on SUBDUE's result.

Deliverable 6: Starting with the above SUBDUE input file with three positive examples of a benzene ring, add six negative examples. Each negative example will be a benzene ring, but with one of the bonds missing (a different bond missing for each of the six negative examples). Provide your input file and the corresponding SUBDUE output file.

As you will see, the results are the same; SUBDUE again learns the correct structure.

Deliverable 7: Provide a discussion of your experience learning the benzene-ring concept with PROGOL and SUBDUE in terms the differences in representations, inputs, outputs, and performance.

References

1. PROGOL Logic-based Relational Learner. (http://www.doc.ic.ac.uk/~shm/progol.html)
2. SUBDUE Graph-based Relational Learner. (http://www.subdue.org)
3. D. Cook and L. Holder. Graph-Based Data Mining. *IEEE Intelligent Systems*, 15(2), pages 32-41, 2000. (http://www.subdue.org/papers/CookIEEE-IS00.pdf)
4. S. Dzeroski and N. Lavrac (editors). Relational Data Mining. Springer, Berlin, 2001.
5. L. Holder and D. Cook. Graph-Based Relational Learning: Current and Future Directions. *ACM SIGKDD Explorations*, Volume 5, Issue 1, July 2003. (http://www.subdue.org/papers/HolderSIGKDDExplorations03.pdf)
6. N. Lavrac and S. Dzeroski. Inductive Logic Programming: Techniques and Applications. Ellis Horwood, New York, 1994.
7. S. Muggleton. Inverse Entailment and Progol. *New Generation Computing Journal*, Vol. 13, pp. 245-286, 1995. (http://www.doc.ic.ac.uk/~shm/Papers/InvEnt.ps.gz)
8. I. Witten and E. Frank. Data Mining: Practical Machine Learning Tools and Techniques (Second Edition), Morgan Kaufmann, 2005. (http://www.cs.waikato.ac.nz/~ml/weka/book.html)